RAIRO-Theor. Inf. Appl. 55 (2021) 6
https://doi.org/10.1051/ita/2021006

RAIRO - Theoretical Informatics and Applications
www.rairo-ita.org

DIGGING INPUT-DRIVEN PUSHDOWN AUTOMATA

Martin Kutrib[*] and Andreas Malcher

Abstract. Input-driven pushdown automata (IDPDA) are pushdown automata where the next action on the pushdown store (push, pop, nothing) is solely governed by the input symbol. Nowadays such devices are usually defined such that popping from the empty pushdown does not block the computation but continues it with empty pushdown. Here, we consider IDPDAs that have a more balanced behavior concerning pushing and popping. Digging input-driven pushdown automata (DIDPDA) are basically IDPDAs that, when forced to pop from the empty pushdown, dig a hole of the shape of the popped symbol in the bottom of the pushdown. Popping further symbols from a pushdown having a hole at the bottom deepens the current hole furthermore. The hole can only be filled up by pushing symbols previously popped. We study the impact of the new behavior of DIDPDAs on their power and compare their capacities with the capacities of ordinary IDPDAs and tinput-driven pushdown automata which are basically IDPDAs whose input may be preprocessed by length-preserving finite state transducers. It turns out that the capabilities are incomparable. We address the determinization of DIDPDAs and their descriptional complexity, closure properties, and decidability questions.

Mathematics Subject Classification. 68Q45, 68Q15.

Received December 6, 2019. Accepted March 25, 2021.

1. Introduction

In connection with an upper bound for the space needed for the recognition of deterministic context-free languages, a subclass of pushdown automata has been introduced in [20]. These so-called *input-driven* pushdown automata (IDPDA) work in real time and, more importantly, in an input-driven way. That is, no moves on empty input are allowed, and the actions on the pushdown store are dictated by the input symbols. To this end, the input alphabet is partitioned into three subsets, where one subset contains symbols on which the automaton pushes a symbol onto the pushdown store, one subset contains symbols on which the automaton pops a symbol, and one subset contains symbols on which the automaton leaves the pushdown unchanged and makes a state change only. The results in [20] and the follow-up papers [7, 24] comprise the equivalence of nondeterministic and deterministic models and the proof that the membership problem is solvable in logarithmic space.

The investigation of input-driven pushdown automata has been revisited in [1, 2], where such devices are called visibly pushdown automata or nested word automata. Some of the results are descriptional complexity aspects for the determinization as well as closure properties and decidability questions which turned out to be similar to those of finite automata. Further aspects such as the minimization of IDPDAs and a

Keywords and phrases: Input-driven pushdown automata, empty pushdown behavior, computational capacity, determinization, descriptional complexity, closure properties, decidability problems.

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany.

[*] Corresponding author: kutrib@informatik.uni-giessen.de

comparison with other subfamilies of deterministic context-free languages have been studied in [5, 6]. A recent survey with many valuable references on complexity aspects of input-driven pushdown automata may be found in [22].

The properties and features of IDPDAs revived the research on input-driven pushdown languages and triggered the study of further input-driven automata types, such as input-driven variants of, for example, multi-pushdown automata or more general auxiliary storages [14, 18], (ordered) multi-pushdown automata [4], scope-bounded multi-pushdown automata [16], stack automata [3], queue automata [12], etc. In particular, input-driven ordered pushdown automata obey the limitation that a pushdown can be popped only if all the lower indexed pushdowns are empty. Nevertheless, even the variant with two pushdowns accepts non-context-free languages as crossing dependencies and cannot be determinized [15].

However, the questions asked in connection with IDPDAs have widened since the early papers from the eighties. Additionally, the definition of input-driven to visibly pushdown automata has changed at a certain point. In the early papers, IDPDAs are defined as ordinary real-time pushdown automata whose behavior on the pushdown is solely driven by the input symbols. The problem that an IDPDA could be forced to pop from the empty stack has not been addressed. So, following a common sense one could say that in such a situation the computation blocks. Later [1], popping from the empty pushdown was simply allowed by defining that popping from the empty pushdown results in an empty pushdown and the computation may continue. While it seems to be natural to overcome the problem by defining some action in such situations and to continue the computation, it is somehow unbalanced and, thus, artificial simple to say that the pushdown remains empty. In this way, in fact, on every input symbol that requires a push operation some additional symbol is pushed, but not for every input symbol that requires a pop operation some symbol is popped. So, an interesting question is to what extent the definition gives additional power to the devices or whether it restricts the capacity.

Here, we consider input-driven pushdown automata that behave differently when they have to pop from the empty pushdown. In order to implement a more balanced behavior, one can imagine that popping a specific symbol from the empty pushdown digs a hole in the bottom of the pushdown, where the hole has the shape of the symbol. This means that the hole can only be filled up by pushing that symbol again. Popping further symbols from a pushdown having a hole at the bottom deepens the current hole furthermore. As usual, for a transition the digging input-driven pushdown automaton looks at the pushdown from above. If the pushdown has a hole, the automaton 'sees' the bottom of the hole, that is, the (shape of the) symbol which was popped as last. These devices are called *digging input-driven pushdown automata* (DIDPDA) and are studied in the sequel. In particular, the basic and formal definition is given and clarified by an introductory example in the next section. In Section 3, the computational capacity of DIDPDAs is studied. Of particular interest is the impact of the new behavior of DIDPDAs on their power. So, among others the capacities are compared with the capacities of ordinary IDPDAs and so-called *tinput-driven pushdown automata* which are basically IDPDAs whose input may be preprocessed by length-preserving finite state transducers [13]. It turns out that the family of languages accepted by DIDPDAs is incomparable with the other devices. So, digging may give power to the machines and, on the other hand, allowing to pop from the empty pushdown without getting stuck can be utilized to perform computations that are impossible for machines that actually *have* to perform the action required by the input symbol. Section 4 considers the determinization of DIDPDAs and its descriptional complexity, where upper and lower bounds on the size are given that match in the order of magnitude of the second exponent. Closure properties and decidability questions for digging pushdown automata are studied in Section 5. We distinguish here in particular the important special case that all automata involved share the same partition of the input alphabet [1] from the general one. Finally, in Section 6 we consider the family of all languages that are either accepted by DIDPDAs or by IDPDAs and we study again the closure properties and decidability questions for this new family. We would like to note that a preliminary version of this work was presented at the 11th Workshop on Non-Classical Models of Automata and Applications (NCMA 2019), Valencia, Spain, July 2–3, 2019, and it is published in [11].

2. Preliminaries

We denote the non-negative integers $\{0, 1, 2, \dots\}$ by $\mathbb{N}$. Let Σ^* denote the set of all words over the finite alphabet Σ. The *empty word* is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The set of words of length at most $n \geq 0$ is denoted by $\Sigma^{\leq n}$. The *reversal* of a word w is denoted by w^R. For the *length* of w we write $|w|$. For the number of occurrences of a symbol a in w the notation $|w|_a$ is used. We use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*. We write 2^S for the power set and $|S|$ for the cardinality of a set S. We say that two language families $\mathscr{L}_1$ and $\mathscr{L}_2$ are *incomparable* if $\mathscr{L}_1$ is not a subset of $\mathscr{L}_2$ and vice versa.

A classical deterministic pushdown automaton (DPDA) is called real-time, if no moves on empty input are allowed. A DPDA is called input-driven (IDPDA) if the next input symbol defines the next action on the pushdown store, that is, pushing a symbol onto the pushdown store, popping a symbol from the pushdown store, or changing the state without modifying the pushdown store. To this end, the input alphabet Σ is partitioned into the sets Σ_D, Σ_R, and Σ_N, that control the actions push (D), pop (R), and state change only (N). However, if the next input symbol forces the IDPDA to pop a symbol from the empty pushdown then the computation does not get stuck but continues with an empty pushdown. At this point, a digging input-driven pushdown automaton has the different behavior mentioned above. In order to implement this behavior, a copy of the pushdown alphabet is used to represent the symbols below the surface. Moreover, when the transition function pops from the pushdown, it explicitly gives the symbol to be popped. A formal definition is:

Definition 2.1. A *deterministic digging input-driven pushdown automaton*, abbreviated as DIDPDA, is a system $M = \langle Q, \Sigma, \Gamma, q_0, F, \perp, \delta_D, \delta_R, \delta_N \rangle$, where

1. Q is the finite set of *internal states*,
2. Σ is the finite set of *input symbols* partitioned into the disjoint sets Σ_D, Σ_R, and Σ_N,
3. Γ is the finite set of *pushdown symbols*, where $\mathring{\Gamma} = \{ \mathring{X} \mid X \in \Gamma \}$ is a copy of Γ,
4. $q_0 \in Q$ is the *initial state*,
5. $F \subseteq Q$ is the set of *accepting states*,
6. $\perp \notin \Gamma$ is the *empty pushdown symbol*,
7. $\delta_D \colon Q \times \Sigma_D \times (\Gamma \cup \{\perp\} \cup \mathring{\Gamma}) \to Q \times \{ \mathtt{push}(x) \mid x \in \Gamma \}$ is a partial transition function, where for $\mathring{g} \in \mathring{\Gamma}$ there are no transitions $\delta_D(p, a, \mathring{g}) = (q, \mathtt{push}(h))$ with $h \neq g$,
8. $\delta_R \colon Q \times \Sigma_R \times (\Gamma \cup \{\perp\} \cup \mathring{\Gamma}) \to Q \times \{ \mathtt{pop}(x) \mid x \in \Gamma \}$ is a partial transition function, where for $g \in \Gamma$ there are no transitions $\delta_R(p, a, g) = (q, \mathtt{pop}(h))$ with $h \neq g$,
9. $\delta_N \colon Q \times \Sigma_N \times (\Gamma \cup \{\perp\} \cup \mathring{\Gamma}) \to Q$ is a partial transition function.

A *configuration* of a DIDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \perp, \delta_D, \delta_R, \delta_N \rangle$ is a triple (q, w, s), where $q \in Q$ is the current state, $w \in \Sigma^*$ is the unread part of the input, and $s \in \Gamma^* \cup \mathring{\Gamma}^*$ denotes the current pushdown content. If $s \in \Gamma^*$ then we have an ordinary pushdown content, where the leftmost symbol of s is at the top of the pushdown store. If $s \in \mathring{\Gamma}^*$ then we have a pushdown content below the surface, where the rightmost symbol of s at the bottom of the pushdown store is seen by M.

The *initial configuration* for an input string w is set to (q_0, w, λ). During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$.

Let $a \in \Sigma$, $w \in \Sigma^*$ and $s \in \Gamma^* \cup \mathring{\Gamma}^*$. We set

1. $(q, aw, zs) \vdash (q', w, z'zs)$, if $a \in \Sigma_D$, $z \in \Gamma$, and $(q', \mathtt{push}(z')) = \delta_D(q, a, z)$,
2. $(q, aw, \lambda) \vdash (q', w, z')$, if $a \in \Sigma_D$ and $(q', \mathtt{push}(z')) = \delta_D(q, a, \perp)$,
3. $(q, aw, s\mathring{z}) \vdash (q', w, s)$, if $a \in \Sigma_D$, $\mathring{z} \in \mathring{\Gamma}$, and $(q', \mathtt{push}(z)) = \delta_D(q, a, \mathring{z})$,
4. $(q, aw, zs) \vdash (q', w, s)$, if $a \in \Sigma_R$, $z \in \Gamma$, and $(q', \mathtt{pop}(z)) = \delta_R(q, a, z)$,
5. $(q, aw, \lambda) \vdash (q', w, \mathring{z})$, if $a \in \Sigma_R$ and $(q', \mathtt{pop}(z)) = \delta_R(q, a, \perp)$,
6. $(q, aw, s\mathring{z}) \vdash (q', w, s\mathring{z}\mathring{z}')$, if $a \in \Sigma_R$, $\mathring{z} \in \mathring{\Gamma}$, and $(q', \mathtt{pop}(z')) = \delta_R(q, a, \mathring{z})$,
7. $(q, aw, zs) \vdash (q', w, zs)$, if $a \in \Sigma_N$, $z \in \Gamma$, and $q' = \delta_N(q, a, z)$,
8. $(q, aw, \lambda) \vdash (q', w, \lambda)$, if $a \in \Sigma_N$ and $q' = \delta_N(q, a, \perp)$.

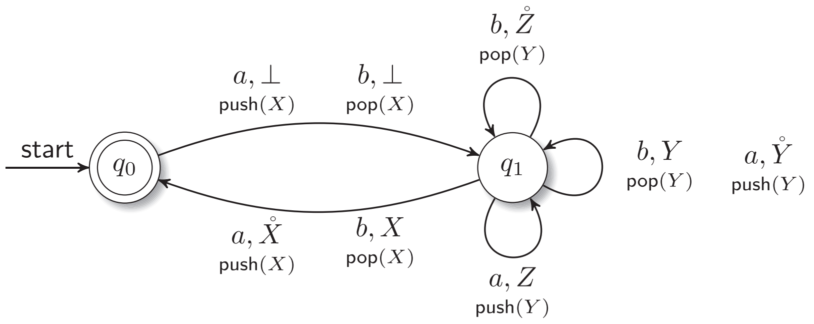

FIGURE 1. A deterministic digging input-driven pushdown automaton accepting the language $\{ w \in \{a,b\}^* \mid |w|_a = |w|_b \}$. Edges corresponding to transitions $\delta(p,a,g) = (q, \mathsf{op}(h))$ are labeled a, g on the first line and $\mathsf{op}(h)$ on the second line of the label. Symbol Z may be X or Y.

9. $(q, aw, s\mathring{z}) \vdash (q', w, s\mathring{z})$, if $a \in \Sigma_N$, $\mathring{z} \in \mathring{\Gamma}$, and $q' = \delta_N(q, a, \mathring{z})$.

So, whenever the pushdown store is empty, the successor configuration is computed by the transition functions with the special empty pushdown symbol $\bot$. Note that $s \in \Gamma^* \cup \mathring{\Gamma}^*$, that is, if one symbol of s belongs to Γ then all symbols of s belong to Γ and, similarly for $\mathring{\Gamma}$. As usual, we define the reflexive and transitive closure of $\vdash$ by $\vdash^*$. The language accepted by the DIDPDA M is the set $L(M)$ of words for which there exists some computation beginning in the initial configuration and ending in a configuration in which the whole input is read and an accepting state is entered. Formally:

$$L(M) = \{ w \in \Sigma^* \mid (q_0, w, \lambda) \vdash^* (q, \lambda, s) \text{ with } q \in F, s \in \Gamma^* \cup \mathring{\Gamma}^* \}.$$

In general, the family of all languages accepted by an automaton of some type X will be denoted by $\mathscr{L}(X)$. In order to clarify this notion, we continue with an example.

Example 2.2. The language $L = \{ w \in \{a,b\}^* \mid |w|_a = |w|_b \}$ is accepted by the DIDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ with the state set $Q = \{q_0, q_1\}$, $\Sigma_D = \{a\}$, $\Sigma_R = \{b\}$, $\Sigma_N = \emptyset$, the set of pushdown symbols $\Gamma = \{X, Y\}$, the set of final states $F = \{q_0\}$, and the transition functions specified as:

$$\begin{array}{rlcl rlcl}
(1) & \delta_D(q_0, a, \bot) & = & (q_1, \texttt{push}(X)) & (6) & \delta_R(q_0, b, \bot) & = & (q_1, \texttt{pop}(X)) \\
(2) & \delta_D(q_1, a, X) & = & (q_1, \texttt{push}(Y)) & (7) & \delta_R(q_1, b, X) & = & (q_0, \texttt{pop}(X)) \\
(3) & \delta_D(q_1, a, Y) & = & (q_1, \texttt{push}(Y)) & (8) & \delta_R(q_1, b, Y) & = & (q_1, \texttt{pop}(Y)) \\
(4) & \delta_D(q_1, a, \mathring{X}) & = & (q_0, \texttt{push}(X)) & (9) & \delta_R(q_1, b, \mathring{X}) & = & (q_1, \texttt{pop}(Y)) \\
(5) & \delta_D(q_1, a, \mathring{Y}) & = & (q_1, \texttt{push}(Y)) & (10) & \delta_R(q_1, b, \mathring{Y}) & = & (q_1, \texttt{pop}(Y))
\end{array}$$

The DIDPDA M is depicted in Figure 1.

The basic idea of the construction is as follows. State q_0 is entered whenever the pushdown gets empty. In order to detect that this will happen after the next step, the first symbol pushed onto the pushdown is X while further symbols on top of X are Y's (Transitions (1), (2), (3)). Similarly, if the pushdown content falls below the surface, the first symbol popped is an X, while further symbols below X are Y's (Transitions (6), (9), (10)). In this way, after popping an X from above the surface or pushing an X under the surface, the pushdown gets empty and M enters state q_0 (Transitions (4), (7)). Since there is a push operation for any input symbol a and a pop operation for any input symbol b, the input string belongs to L, if and only if after its processing the pushdown is empty, that is, M enters state q_0.

We note that the language of the previous example can be accepted by finite state models with unconventional reading order as well. For example, it can be accepted by finite automata with translucent letters [21] or by jumping finite automata [19].

3. Computational capacity

Any language accepted by some deterministic or nondeterministic IDPDA is accepted by a real-time deterministic pushdown automaton as well. On the other hand, for example, the language $\{ a^n\$a^n \mid n \geq 0 \}$ is not accepted by any IDPDA. This immediately implies the fact that the language family accepted by IDPDAs is a proper subset of the real-time deterministic context-free languages. Here, we start to explore the relationships between DIDPDA and other types of devices by presenting a result that yields a construction that effectively converts any given DIDPDA into an equivalent real-time deterministic pushdown automaton.

Theorem 3.1. *The family $\mathscr{L}$(DIDPDA) is included in the family of real-time deterministic context-free languages.*

Proof. Let $M = \langle Q, \Sigma, \Gamma, q_0, F, \perp, \delta_D, \delta_R, \delta_N \rangle$ be an arbitrary DIDPDA. We will construct an equivalent real-time DPDA $M' = \langle Q', \Sigma, \Gamma, q_0', F', \perp, \delta' \rangle$ as follows.

Basically, M' simulates M where it remembers in its states whether the pushdown of M' is above or below the surface. Since M' can freely push or pop symbols not depending on the current input symbol, a pushdown below the surface is simulated by pushing onto instead of popping from the empty pushdown. To implement this behavior, we define $Q' = Q \times \{\uparrow, \downarrow\}$, $q_0' = (q_0, \uparrow)$, and $F' = F \times \{\uparrow, \downarrow\}$.

The transition function δ' is defined as follows for $q, q' \in Q$, $a \in \Sigma$, $z \in \Gamma \cup \{\perp\}$, and $z' \in \Gamma$. Let $\mathring{\perp} = \perp$, then

$$\delta'((q,\uparrow), a, z) = ((q',\uparrow), \texttt{push}(z')) \text{ if } \delta_D(q,a,z) = (q', \texttt{push}(z')) \text{ and } a \in \Sigma_D,$$

$$\delta'((q,\uparrow), a, z) = ((q',\uparrow), \texttt{pop}(z)) \text{ if } \delta_R(q,a,z) = (q', \texttt{pop}(z)),\ a \in \Sigma_R \text{ and } z \neq \perp,$$

$$\delta'((q,\uparrow), a, \perp) = ((q',\downarrow), \texttt{push}(z')) \text{ if } \delta_R(q,a,\perp) = (q', \texttt{pop}(z')) \text{ and } a \in \Sigma_R,$$

$$\delta'((q,\downarrow), a, z) = ((q',\downarrow), \texttt{pop}(z)) \text{ if } \delta_D(q,a,\mathring{z}) = (q', \texttt{push}(z)),\ a \in \Sigma_D \text{ and } z \neq \perp,$$

$$\delta'((q,\downarrow), a, \perp) = ((q',\uparrow), \texttt{push}(z')) \text{ if } \delta_D(q,a,\perp) = (q', \texttt{push}(z')) \text{ and } a \in \Sigma_D,$$

$$\delta'((q,\downarrow), a, z) = ((q',\downarrow), \texttt{push}(z')) \text{ if } \delta_R(q,a,\mathring{z}) = (q', \texttt{pop}(z')) \text{ and } a \in \Sigma_R,$$

$$\delta'((q,\uparrow), a, z) = (q',\uparrow) \text{ if } \delta_N(q,a,z) = q' \text{ and } a \in \Sigma_N,$$

$$\delta'((q,\downarrow), a, z) = (q',\downarrow) \text{ if } \delta_N(q,a,\mathring{z}) = q' \text{ and } a \in \Sigma_N.$$

By construction, a word w is accepted by M if and only if w is accepted by M'. Inspecting δ' shows that M' is indeed a deterministic pushdown automaton working in real time. □

Since also DIDPDAs either always have to push or always have to pop when reading the a's, the language $\{ a^n\$a^n \mid n \geq 0 \}$ from above is clearly not accepted by any DIDPDA. So, the inclusion of Theorem 3.1 is strict.

Corollary 3.2. *The family $\mathscr{L}$(DIDPDA) is a proper subset of the family of real-time deterministic context-free languages.*

In [13], input-driven automata are extended in such a way that the input is preprocessed by a deterministic injective and length-preserving finite state transducer. The corresponding devices are called *tinput-driven pushdown automata* (TDPDA). In this way, for example, language $\{ a^n\$a^n \mid n \geq 0 \}$ is accepted by such a device. In general, it turned out that TDPDAs are strictly more powerful than IDPDAs but still not as powerful as real-time deterministic pushdown automata. It is shown that, for example, the language $\{ a^nb^{n+m}a^m \mid n, m \geq 1 \}$ is not accepted by any TDPDA. In order to compare the computational capacities of IDPDAs and TDPDAs with

DIDPDAs, we next show that this language is accepted by some deterministic digging input-driven pushdown automaton.

Lemma 3.3. *There is a language accepted by some DIDPDA that cannot be accepted by any TDPDA and, thus, cannot be accepted by any IDPDA.*

Proof. We use $L = \{\, a^n b^{n+m} a^m \mid n, m \geq 1 \,\}$ as a witness language. It is known [13] that L does not belong to $\mathscr{L}(\text{TDPDA})$. Since $\mathscr{L}(\text{IDPDA}) \subset \mathscr{L}(\text{TDPDA})$, L cannot be accepted by any IDPDA.

On the other hand, by Example 2.2, there is some DIDPDA accepting the language $L' = \{\, w \in \{a,b\}^* \mid |w|_a = |w|_b \,\}$. By simulating a deterministic finite automaton in parallel to the actual computation, it can immediately be seen that the family $\mathscr{L}(\text{DIDPDA})$ is closed under intersection with regular languages. Therefore, $L = L' \cap a^+b^+a^+$ belongs to $\mathscr{L}(\text{DIDPDA})$ as well. $\square$

So, compared with IDPDAs that may arbitrarily pop from the empty pushdown, digging may give power to the machines. On the other hand, allowing to pop from the empty pushdown without getting stuck can be utilized to perform computations that are impossible for machines that actually have to perform the action required by the input symbol.

Lemma 3.4. *There is a language accepted by some IDPDA and, thus, by some TDPDA, that cannot be accepted by any DIDPDA.*

Proof. We use the language $L = \{\, a^k b^\ell a^m b^n \mid 1 \leq k < \ell \text{ and } 1 \leq m < n \,\}$ as a witness language.

An IDPDA accepting the language L pushes upon reading a's and pops upon reading b's. Then, it has to check whether the input format is correct, and whether it pops at least once from the empty pushdown while reading the sequences of b's. So, L belongs to the family $\mathscr{L}(\text{IDPDA})$. Since $\mathscr{L}(\text{IDPDA}) \subset \mathscr{L}(\text{TDPDA})$, L is accepted by some TDPDA as well.

Now, we assume that some DIDPDA $M = \langle Q, \{a,b\}, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ is accepting language L. If at least one of Σ_D and Σ_R is empty, M accepts a regular language. Since L is non-regular, we have to consider the two cases $\Sigma_N = \emptyset$ and either $\Sigma_D = \{a\}$, $\Sigma_R = \{b\}$, or $\Sigma_D = \{b\}$, $\Sigma_R = \{a\}$.

So, let $\Sigma_D = \{a\}$, $\Sigma_R = \{b\}$, and $\Sigma_N = \emptyset$. First we choose some arbitrary $k \geq 1$. Let $(q_0, a^k b^k, \lambda) \vdash^{2k} (q_1, \lambda, \lambda)$ be the computation of M on input $a^k b^k$. Continuing the computation on factor b^i with $i > |Q| \cdot |\Gamma|$ large enough will drive M into a cycle of length $c \geq 1$, possibly after an initial part of length $0 \leq c_0 \leq |Q| \cdot |\Gamma|$ as follows. For all $j_1 \geq 1$:

$$\begin{aligned}(q_1, b^i, \lambda) &\vdash^{c_0} (q_2, b^{i-c_0}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1) \vdash^{c} (q_2, b^{i-c_0-c}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)\\ &\vdash^{(j_1-1)\cdot c} (q_2, b^{i-c_0-j_1\cdot c}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{j_1}).\end{aligned}$$

Next, we set $j_1 = (c+1)\cdot|Q|$, $\ell = k + c_0 + j_1 \cdot c$, $m = c \cdot |Q|$, and consider the accepting computation of M on input $a^k b^\ell a^m b^{m+1}$:

$$\begin{aligned}(q_0, a^k b^\ell a^m b^{m+1}, \lambda) &\vdash^{2k} (q_1, b^{c_0+j_1\cdot c} a^m b^{m+1}, \lambda)\\ &\vdash^{c_0} (q_2, b^{j_1\cdot c} a^m b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1)\\ &\vdash^{j_1\cdot c} (q_2, a^m b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{j_1}).\end{aligned}$$

Since for the continuation of the computation on the following factor a^m there are at most $c \cdot |Q|$ different possibilities for being in a state and seeing a pushdown symbol, and $m = c \cdot |Q|$, such a situation appears at least twice while processing a^m, say with state q_3 and pushdown symbol $\mathring{z}_r$, for some $1 \leq r \leq c$. In general, in this phase M runs again through a cycle, say of length $\hat{c} \geq 1$, possibly after an initial part of length $\hat{c}_0$, where where $0 \leq \hat{c}_0 < \hat{c}_0 + \hat{c} \leq c \cdot |Q| = m$. That is, for some $0 \leq s_1 < s_2 < |Q|$, $q_f \in F$, and $q_4 \in Q$ the computation continues as

$$\begin{aligned}
&(q_2, a^m b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{j_1})\\
&\vdash^{\hat{c}_0} (q_3, a^{m-\hat{c}_0} b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{(c+1)\cdot|Q|-s_1}\mathring{z}_2\cdots\mathring{z}_r)\\
&\vdash^{\hat{c}} (q_3, a^{m-\hat{c}_0-\hat{c}} b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{(c+1)\cdot|Q|-s_2}\mathring{z}_2\cdots\mathring{z}_r)\\
&\vdash^{m-\hat{c}_0-\hat{c}} (q_4, b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{c\cdot|Q|})\\
&\vdash^{m+1} (q_f, \lambda, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{c\cdot|Q|}\mathring{x}_1\mathring{x}_2\cdots\mathring{x}_{m+1}).
\end{aligned}$$

Now, we consider the computation of M on input $a^k b^\ell a^{m+c\cdot\hat{c}} b^{m+1}$:

$$\begin{aligned}
&(q_0, a^k b^\ell a^{m+c\cdot\hat{c}} b^{m+1}, \lambda)\\
&\vdash^{k+\ell} (q_2, a^{m+c\cdot\hat{c}} b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{j_1})\\
&\vdash^{\hat{c}_0+\hat{c}} (q_3, a^{m+c\cdot\hat{c}-\hat{c}_0-\hat{c}} b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{(c+1)\cdot|Q|-s_2}\mathring{z}_2\cdots\mathring{z}_r)\\
&\vdash^{c\cdot\hat{c}} (q_3, a^{m-\hat{c}_0-\hat{c}} b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{(c+1)\cdot|Q|-s_2-\hat{c}}\mathring{z}_2\cdots\mathring{z}_r)\\
&\vdash^{m-\hat{c}_0-\hat{c}} (q_4, b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{c\cdot|Q|-\hat{c}}).
\end{aligned}$$

Since $\hat{c} \leq c\cdot|Q|$, the computation ends accepting:

$$\begin{aligned}
&(q_4, b^{m+1}, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{c\cdot|Q|-\hat{c}})\\
&\vdash^{m+1} (q_f, \lambda, \mathring{y}_1\mathring{y}_2\cdots\mathring{y}_{c_0-1}\mathring{z}_1(\mathring{z}_2\cdots\mathring{z}_c\mathring{z}_1)^{c\cdot|Q|-\hat{c}}\mathring{x}_1\mathring{x}_2\cdots\mathring{x}_{m+1}).
\end{aligned}$$

So, the input $a^k b^\ell a^{m+c\cdot\hat{c}} b^{m+1}$ is accepted, but does not belong to L, since $m + c\cdot\hat{c} \not< m+1$, a contradiction.

Finally, the case $\Sigma_D = \{b\}$, $\Sigma_R = \{a\}$, and $\Sigma_N = \emptyset$ has to be considered. The contradiction in this case follows almost literally as in the first case, with the roles played by pushing and popping and symbols from $\mathring{\Gamma}$ and Γ interchanged. □

By the previous lemmas we obtain the following incomparabilities.

Theorem 3.5. *The family $\mathscr{L}(DIDPDA)$ is incomparable with the family $\mathscr{L}(IDPDA)$ and the family $\mathscr{L}(TDPDA)$.*

4. Determinization

It is well known that nondeterministic IDPDAs can be determinized. Okhotin and Salomaa [22] traced this result back to [24]. They give a clear proof that shows that 2^{n^2} states are sufficient to simulate an n-state nondeterministic IDPDA by a deterministic one. However, in [22] and [1] IDPDAs are considered in a certain normal form. That is, neither the push nor the state change only operations depend on the topmost pushdown symbol. Since any deterministic pushdown automaton and any deterministic input-driven pushdown automaton can be converted to this normal form, the general computational capacity does not change. But by this conversion the number of states changes since the topmost pushdown symbol has to be remembered in the states. Thus, when we compare such automata with the original definition of IDPDAs in [24], that is based on the usual definition of pushdown automata and does not require a normal form, we obtain that the state complexity bound of 2^{n^2} achieved for the determinization of IDPDAs in normal form is lower than in general. Moreover, results in [8, 9] show that states can be traded for pushdown symbols and vice versa. Hence, the size of a pushdown automaton is affected by the size of its state set as well as by the number of pushdown symbols. Since here we are closer to the original definition, the size of a digging input-driven pushdown automaton is measured not only by its states but as the product of the number of states and the number of pushdown symbols. Accordingly, we define the function size that maps a digging input-driven pushdown automaton to its size.

Definition 4.1. A *nondeterministic digging input-driven pushdown automaton*, abbreviated as NDIDPDA, is a system $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N\rangle$, where Q, Σ, Γ, q_0, and F are defined as for DIDPDAs. The transition functions are now nondeterministic, that is,

$$\begin{aligned}
&\delta_D\colon Q \times \Sigma_D \times (\Gamma \cup \{\bot\} \cup \mathring{\Gamma}) \to 2^{Q\times\{\,\mathtt{push}(x) \mid x\in\Gamma\,\}},\\
&\delta_R\colon Q \times \Sigma_R \times (\Gamma \cup \{\bot\} \cup \mathring{\Gamma}) \to 2^{Q\times\{\,\mathtt{pop}(x) \mid x\in\Gamma\,\}}, \text{ and}\\
&\delta_N\colon Q \times \Sigma_N \times (\Gamma \cup \{\bot\} \cup \mathring{\Gamma}) \to 2^{Q},
\end{aligned}$$

where the restrictions from the definition of DIDPDAs are adapted. As usual, an input is accepted if there is an accepting computation on it.

The next theorem shows the determinization of NDIDPDAs. Basically, the idea of the proof is along the lines of [22].

Theorem 4.2. *Let $n \geq 1$ and M be an n-state nondeterministic digging input-driven pushdown automaton with input alphabet $\Sigma = \Sigma_D \cup \Sigma_R \cup \Sigma_N$ and set of pushdown symbols Γ. Then, an equivalent DIDPDA with at most $2^{n^2(|\Gamma|+1)}$ states and $2^{n^2(|\Gamma|+1)}(|\Sigma_D| + |\Sigma_R|)$ pushdown symbols can effectively be constructed.*

Proof. Let $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N\rangle$ be a nondeterministic digging input-driven pushdown automaton. Then we can construct an equivalent DIDPDA $M' = \langle Q', \Sigma, \Gamma', q_0', F', \bot, \delta_D', \delta_R', \delta_N'\rangle$ as follows. We set

$$\begin{aligned}
Q' &= 2^{Q\times Q\times(\Gamma\cup\{\bot\})},\\
q_0' &= \{(q_0, q_0, \bot)\}, \text{ and}\\
F' &= \{\, P \in Q' \mid \text{ there is } (q, r, z) \in P \text{ where } r \in F \,\}.
\end{aligned}$$

The states of M' have the following interpretation. Each triple (p, q, z) says either that if M enters state p on pushing z on a pushdown whose content is above the surface then state q is reachable when M sees this z on top of the pushdown again, or that if M enters state p on popping z from a pushdown whose content is below the surface then state q is reachable when M sees this $\mathring{z}$ on the bottom of the pushdown again.

Since M is nondeterministic, the states of M' are subsets of such triples. So, the initial state of M' consists of the triple $(q_0, q_0, \bot)$, where state q_0 is reached from state q_0 with empty pushdown simply by doing nothing.

During the computation, M' pushes its current state together with the current input symbol whenever the current input symbol enforces a push (of M as well as of M') to a pushdown whose content is above the surface. Similarly, M' pops its current state together with the current input symbol whenever the current input symbol enforces a pop (of M as well as of M') from a pushdown whose content is below the surface. So, we will define the set of pushdown symbols as $\Gamma' = 2^{Q\times Q\times(\Gamma\cup\{\bot\})} \times (\Sigma_D \cup \Sigma_R)$.

Next, the transition functions have to be defined in order to enable this behavior.

Let $a \in \Sigma_D$ be the current input symbol, P be the current state of M', and $(Z, x) \in \Gamma'$. Then we define $\delta_D'(P, a, (Z, x)) = (P', \mathtt{push}((P, a)))$, where

$$P' = \{\, (q', q', z') \mid \text{ there is } (q, r, z) \in P \text{ such that } (q', z') \in \delta_D(r, a, z) \,\}.$$

If $(\mathring{Z}, x) \in \mathring{\Gamma}'$ then we define $\delta_D'(P, a, (\mathring{Z}, x)) = (P', \mathtt{push}((Z, x)))$, where

$$\begin{aligned}
P' = \{\, (q, q'', z) \mid \text{ there is } (q, r, z) \in Z \text{ and } (q', r', z') \in P \text{ such that}\\
(q', \mathtt{pop}(z')) \in \delta_R(r, x, z) \text{ and } (q'', \mathtt{push}(z')) \in \delta_D(r', a, z') \,\}.
\end{aligned}$$

So, in the first case, M' pushes the current context of the simulation together with the current input symbol, and starts to compute the reachable states with the new top of pushdown symbol of M. In the second case, the context of the simulation popped from the pushdown at last (which is seen by M' now) is combined with the simulation that started after that popping (which is given by the state of M'). If both computation paths fit together they are used to define the new triples of the new state of M'.

Now let $a \in \Sigma_N$ be the current input symbol, P be the current state of M', and $(Z,x) \in \Gamma'$. Then we define $\delta'_N(P,a,(Z,x)) = P'$, where

$$P' = \{\, (q,q',z) \mid \text{ there is } (q,r,z) \in P \text{ such that } q' \in \delta_N(r,a,z)\,\}.$$

The definition for $(\overset{\circ}{Z,x}) \in \mathring{\Gamma}'$ is almost identical: $\delta'_N(P,a,(\overset{\circ}{Z,x})) = P'$, where

$$P' = \{\, (q,q',z) \mid \text{ there is } (q,r,z) \in P \text{ such that } q' \in \delta_N(r,a,\mathring{z})\,\}.$$

So, M' directly simulates one step of M in all currently traced computation paths.

Finally, let $a \in \Sigma_R$ be the current input symbol, P be the current state of M', and $(Z,x) \in \Gamma'$. The definition of δ'_R is similar to the definition of δ'_D. Here the roles played by pushing and popping and pushdown contents above and below the surface are interchanged. We define $\delta'_R(P,a,(Z,x)) = (P', \texttt{pop}((Z,x)))$, where

$$\begin{aligned}P' = \{\, (q,q'',z) \mid \text{ there is } (q,r,z) \in Z \text{ and } (q',r',z') \in P \text{ such that}&\\ (q',\texttt{push}(z')) \in \delta_D(r,x,z) \text{ and } (q'',\texttt{pop}(z')) \in \delta_R(r',a,z')\,\}.&\end{aligned}$$

If $(\overset{\circ}{Z,x}) \in \mathring{\Gamma}'$ then we define $\delta'_R(P,a,(\overset{\circ}{Z,x})) = (P', \texttt{pop}((P,a)))$, where

$$P' = \{\, (q',q',z') \mid \text{ there is } (q,r,z) \in P \text{ such that } (q',z') \in \delta_R(r,a,\mathring{z})\,\}.$$

By the constructions and the set of accepting states, when M' reaches an accepting state then this state contains a triple that says that M can reach an accepting state. Conversely, if some computation path of M ends accepting, it is simulated by M' and, thus, M' accepts as well. □

Given some nondeterministic digging input-driven pushdown automaton, an upper bound for the size of an equivalent DIDPDA can immediately be derived from Theorem 4.2. In order to obtain lower bounds, for $k \geq 1$, we consider witness languages

$$L_k = \{\, (\texttt{\$}\{a,b\}^*)^*\texttt{\$}u(\texttt{\$}\{a,b\}^*)^*\texttt{\#}u \mid u \in \{a,b\}^k\,\}.$$

Lemma 4.3. *For $k \geq 1$, the language L_k is accepted by some NDIDPDA with $2k+4$ states and 2^k pushdown symbols.*

Proof. The following NDIDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \perp, \delta_D, \delta_R, \delta_N \rangle$ accepts L_k.

We set $Q = \{q_0, q_1, \dots, q_{k+1}, p_0, p_1, \dots, p_{k+1}\}$, $\Sigma_D = \{\$\}$, $\Sigma_R = \emptyset$, $\Sigma_N = \{a, b, \#\}$, $\Gamma = \{a, b\}^k$, $F = \{p_{k+1}\}$, and we specify the transition functions for $x, v_1, v_2, \dots, v_k \in \{a, b\}$ as:

$$\begin{array}{rrcl}
(1) & \delta_D(q_0, \$, \bot) & = & \{\, (q_0, \mathtt{push}(v)), (q_1, \mathtt{push}(v)) \mid v \in \{a,b\}^k \,\} \\
(2) & \delta_D(q_0, \$, v) & = & \{(q_0, \mathtt{push}(v)), (q_1, \mathtt{push}(v))\} \text{ if } v \neq \bot \\
(3) & \delta_D(q_{k+1}, \$, v) & = & \{(p_0, \mathtt{push}(v))\} \\
(4) & \delta_D(p_0, \$, v) & = & \{(p_0, \mathtt{push}(v))\} \\
(5) & \delta_N(q_0, x, v) & = & \{q_0\} \\
(6) & \delta_N(q_i, x, v_1 v_2 \cdots v_k) & = & \{q_{i+1}\} \text{ for } 1 \leq i \leq k \text{ if } x = v_i \\
(7) & \delta_N(q_{k+1}, \#, v) & = & \{p_1\} \\
(8) & \delta_N(p_0, x, v) & = & \{p_0\} \\
(9) & \delta_N(p_0, \#, v) & = & \{p_1\} \\
(10) & \delta_N(p_i, x, v_1 v_2 \cdots v_k) & = & \{p_{i+1}\} \text{ for } 1 \leq i \leq k \text{ if } x = v_i
\end{array}$$

The basic idea of the construction is as follows. In the initial step (uniquely determined by state q_0 and empty pushdown), M guesses a factor $v \in \{a, b\}^k$ and pushes it (Transition (1)). Moreover, state q_0 with a non-empty pushdown is used to read the prefix of the input and on some \$ to guess whether the next input factor is u (Transitions (1), (2), (5)). Whenever in this phase a \$ enforces a push operation, the guessed v at the top of the pushdown is pushed again (Transition (2)).

When the guess is that u comes next, the state q_1 is entered. Then, states $q_1, q_2, \dots, q_k$ are used to match the input factor u read with the factor v on top of the pushdown (Transition (6)). If and only if the first k symbols of u match v, then M is in state q_{k+1}. Afterwards the computation continues, if and only if the next input symbol is \$ or #, that is, if and only if $u = v$. If the symbol is #, state p_1 is entered (Transition (7). If the symbol is \$, state p_0 is used to read the remaining input up to the symbol #, where M behaves similarly as for state q_0 (Transitions (4), (8)). When M reaches the # in state p_0, it changes to state p_1 (Transition (9)).

Finally, states $p_1, p_2, \dots, p_k$ are used to match the remaining input suffix read with the factor v on top of the pushdown (Transition (10)). If and only if the first k symbols of the suffix match v then M is in state p_{k+1}. Since p_{k+1} is the sole accepting state and the transition functions are undefined for p_{k+1}, M halts. So, it accepts, if and only if it has read the input entirely, that is, if and only if the suffix equals v and, thus equals u, and therefore the input belongs to L_k. □

Applying the determinization of Theorem 4.2 to the NDIDPDA constructed in the proof of Lemma 4.3 gives an equivalent DIDPDA M accepting L_k with at most $2^{|Q|^2(|\Gamma|+1)}$ states and $2^{|Q|^2(|\Gamma|+1)}(|\Sigma_D| + |\Sigma_R|)$ pushdown symbols.

So, the size of M is

$$\begin{aligned}
2^{|Q|^2(|\Gamma|+1)} \cdot 2^{|Q|^2(|\Gamma|+1)}(|\Sigma_D| + |\Sigma_R|) &\leq 2^{2|Q|^2(|\Gamma|+1)} \cdot |\Sigma| \\
&= 2^{2(2k+4)^2(2^k+1)} \cdot 4 \in 2^{O(k^2)\cdot O(2^k)} = 2^{O(2^{2\log(k)})\cdot O(2^k)} = 2^{O(2^{2\log(k)+k})} \in 2^{2^{O(k)}}.
\end{aligned}$$

The next lemma shows that this upper bound derived from the determinization is tight in the order of the second exponent. We recall that the size of a digging IDPDA is defined as the product of the number of states and the number of pushdown symbols.

Lemma 4.4. *For $k \geq 1$, let M be some DIDPDA accepting language L_k. Then,* $\mathsf{size}(M) \geq 2^{2^k}$.

Proof. Let $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ be a DIDPDA that accepts L_k.

We consider subsets of $\{a, b\}^k$ and for any such $S = \{v_1, v_2, \dots, v_m\} \in 2^{\{a,b\}^k}$, let the ordering of the elements be arbitrarily fixed. Then, a word w_S is defined as $\$v_1\$v_2 \cdots \$v_m\$$.

Assume now that for two different such subsets R and S, automaton M is in the same state and sees the same symbol at the pushdown after processing the words w_R and w_S. Since R and S are different, there is a word u belonging to one set but not to the other, say $u \in R \setminus S$.

Next, we distinguish three cases dependent on the pushdown symbol that M sees after processing w_R and w_S.

If this symbol is $\perp$, the words w_R and w_S are extended by $\#u$. So, $w_R\#u$ is accepted, if and only if $w_S\#u$ is accepted. This is a contradiction since $u \in R \setminus S$ and, thus, $w_R\#u \in L_k$ and $w_S\#u \notin L_k$

In the next case, M sees a symbol from Γ at the pushdown after processing w_R and w_S. This implies that Σ_D is not empty and contains at least one of the symbols a, b, or $\$$. We denote one of these symbols from Σ_D by x and extend w_R and w_S by $x^{k+1}\#u$. The computations on both words are

$$(q_0, w_R x^{k+1}\#u, \perp) \vdash^* (q_1, x^{k+1}\#u, zs) \vdash^* (q_2, \#u, s'zs)$$

and

$$(q_0, w_S x^{k+1}\#u, \perp) \vdash^* (q_1, x^{k+1}\#u, z\hat{s}) \vdash^* (q_2, \#u, s'z\hat{s})$$

where $s' \in \Gamma^{k+1}$, $q_1, q_2 \in Q$, $z \in \Gamma$, and $s, \hat{s} \in \Gamma^*$. Since $|u| = k$, we conclude that the continuations of the computations do not depend on s or $\hat{s}$ at the pushdown. So, $w_R x^{k+1}\#u$ is accepted, if and only if $w_S x^{k+1}\#u$ is accepted. As before, this is a contradiction since $u \in R \setminus S$ and, thus, $w_R x^{k+1}\#u \in L_k$ and $w_S x^{k+1}\#u \notin L_k$.

In the last case, M sees a symbol from $\mathring{\Gamma}$ at the pushdown after processing w_R and w_S. This implies that Σ_R is not empty and contains at least one symbol from the set $\{a, b, \$\}$. As in the case before, we denote one of these symbols from Σ_R by x and extend w_R and w_S by $x^{k+1}\#u$. The computations on both words are now

$$(q_0, w_R x^{k+1}\#u, \perp) \vdash^* (q_1, x^{k+1}\#u, s\mathring{z}) \vdash^* (q_2, \#u, s\mathring{z}s')$$

and

$$(q_0, w_S x^{k+1}\#u, \perp) \vdash^* (q_1, x^{k+1}\#u, \hat{s}\mathring{z}) \vdash^* (q_2, \#u, \hat{s}\mathring{z}s')$$

where $s' \in \mathring{\Gamma}^{k+1}$, $q_1, q_2 \in Q$, $\mathring{z} \in \mathring{\Gamma}$, and $s, \hat{s} \in \mathring{\Gamma}^*$. Since $|u| = k$, we conclude that the continuations of the computations do not depend on s or $\hat{s}$ at the pushdown. So, $w_R x^{k+1}\#u$ is accepted, if and only if $w_S x^{k+1}\#u$ is accepted. As before, this is a contradiction since $u \in R \setminus S$ and, thus, $w_R x^{k+1}\#u \in L_k$ and $w_S x^{k+1}\#u \notin L_k$.

These contradictions show that the assumption that M is in the same state and sees the same symbol at the pushdown for two different subsets R and S is wrong. Since there are 2^{2^k} different subsets, we conclude $|Q| \cdot |\Gamma| \geq 2^{2^k}$. Therefore, the size of M is at least 2^{2^k}. □

5. Closure properties and decidability questions

For input-driven pushdown automata, strong closure properties are shown in [1] *provided that* all automata involved share the same partition of the input alphabet. Here, we distinguish this important special case from the general one. For easier writing, we call the partition of an input alphabet a *signature*, and say that two signatures $\Sigma = \Sigma_D \cup \Sigma_R \cup \Sigma_N$ and $\Sigma' = \Sigma'_D \cup \Sigma'_R \cup \Sigma'_N$ are *compatible*, if and only if

$$\bigcup_{j \in \{D,R,N\}} (\Sigma_j \setminus \Sigma'_j) \cap \Sigma' = \emptyset \quad \text{and} \quad \bigcup_{j \in \{D,R,N\}} (\Sigma'_j \setminus \Sigma_j) \cap \Sigma = \emptyset.$$

Before we start to investigate the closure properties in detail, we state the following preparatory lemma which ensures that the input is completely read in every computation.

Lemma 5.1. *Any DIDPDA can effectively be converted to an equivalent DIDPDA that accepts or rejects its input after having read the input entirely.*

Proof. Let M be a DIDPDA. By definition an input is accepted by M only after having read it entirely. An input is rejected by M if the computation of M ends in a non-accepting state after having read the input entirely, or if the computation of M blocks since the transition function is undefined for the current situation. In the latter case, we have to ensure that the remaining input is processed. So, for the construction of an equivalent DIDPDA M' it is sufficient to add transitions to a new non-accepting state s_- whenever a transition is undefined. Once in state s_-, M' still has to obey the signature: For any input symbol from Σ_N, M' stays in state s_-. For any input symbol $a \in \Sigma_D$, we define $\delta'_D(s_-, a, g) = (s_-, \mathtt{push}(g))$ for $g \in \Gamma$, $\delta'_D(s_-, a, \bot) = (s_-, \mathtt{push}(g))$ for some $g \in \Gamma$, and $\delta'_D(s_-, a, \mathring{g}) = (s_-, \mathtt{push}(g))$ for $\mathring{g} \in \mathring{\Gamma}$. For any input symbol $a \in \Sigma_R$, we define $\delta'_R(s_-, a, g) = (s_-, \mathtt{pop}(g))$ for $g \in \Gamma$, $\delta'_R(s_-, a, \bot) = (s_-, \mathtt{pop}(g))$ for some $g \in \Gamma$, and $\delta'_R(s_-, a, \mathring{g}) = (s_-, \mathtt{pop}(g))$ for $\mathring{g} \in \mathring{\Gamma}$. In this way, now all non-accepting computations of M end in a non-accepting state of M' after having read the input entirely. □

Now, we will first turn to the closure under the Boolean operations.

Lemma 5.2. *Let M and M' be two DIDPDAs with compatible signatures. Then DIDPDAs accepting the intersection $L(M) \cap L(M')$, the union $L(M) \cup L(M')$, and the complement $\overline{L(M)}$ can effectively be constructed.*

Proof. First of all we apply Lemma 5.1 and assume that all DIDPDAs considered read their input entirely before accepting or rejecting. For the closure under intersection we can use the cross-product construction that is known for the closure under intersection for IDPDAs. Basically, both DIDPDAs can be simulated in two tracks of a new DIDPDA having as state set pairs of states and as pushdown symbols pairs of pushdown symbols. Since the signatures are compatible, the push and pop operations of each DIDPDA take place at the same time. Hence, both pushdown stores can be simulated and maintained by one pushdown store. Finally, the input is accepted if both computations simulated are accepting.

For the closure under complementation, we first notice that in every accepting or rejecting computation the input is completely read. Thus, we can construct a new DIDPDA for the complement by interchanging accepting and rejecting states. The closure under complementation also implies the closure under union by applying De Morgan's laws. □

Next, we turn to non-closure results.

Lemma 5.3. *The family $\mathscr{L}(DIDPDA)$ is not closed under concatenation (even with compatible signatures), inverse homomorphism, and length-preserving homomorphism. It is not closed under intersection and union in case of incompatible signatures.*

Proof. The languages $L_1 = \{\, a^n\$a^n \mid n \geq 1 \,\}$ and $L_2 = \{\, a^n\$b^{2n} \mid n \geq 1 \,\}$ are each not accepted by any DIDPDA. On the other hand, $L'_1 = \{\, a^n\$b^n \mid n \geq 1 \,\}$ and $L'_2 = \{\, a^{2n}\$b^{2n} \mid n \geq 1 \,\}$ are each accepted by some DIDPDA. Now, consider the length-preserving homomorphisms $h_1 : \{a, b, \$\} \to \{a, \$\}^+$ mapping both a and b to a and $\$$ to $\$$, and $h_2 : \{a, b, \$\} \to \{a, b, \$\}^+$ mapping a to aa, b to b, and $\$$ to $\$$. Then, $h_1(L'_1) = L_1$ and $h_2^{-1}(L'_2) = L_2$ which implies that $\mathscr{L}$(DIDPDA) is neither closed under length-preserving homomorphism nor under inverse homomorphism.

For the non-closure under intersection consider $L_3 = \{\, a^n b^n c^m \mid n, m \geq 1 \,\}$ and $L_4 = \{\, a^n b^m c^m \mid n, m \geq 1 \,\}$. Both languages can be accepted by DIDPDAs with incompatible signatures. However, $L_3 \cap L_4$ is a non-context-free language which gives the non-closure under intersection. Since $\mathscr{L}$(DIDPDA) is closed under complementation by Lemma 5.2, the non-closure under union can be derived as well.

To obtain the the non-closure under concatenation we consider the language $L_5 = \{\, a^n b^m \mid n, m \geq 1 \text{ and } n < m \,\}$ that is accepted by some DIDPDA. It is known from the proof of Lemma 3.4 that $L_5 \cdot L_5$ is not accepted by any DIDPDA which gives that $\mathscr{L}$(DIDPDA) is not closed under concatenation even with compatible signatures. □

We continue with studying the unary operations Kleene star and reversal. It is known that the family $\mathscr{L}$(IDPDA) is closed under both operations. However, for the reversal the construction for IDPDAs works by considering the *inverted signature* Σ^{-1} where Σ_D and Σ_R are interchanged. This means, that symbols from Σ_R then imply push operations, whereas symbols from Σ_D imply pop operations. If we require that the IDPDA for the reversal has to have the identical signature, then $\mathscr{L}$(IDPDA) is no longer closed under reversal. This can be seen using the language $\{\, a^n b^n \mid n \geq 1 \,\}$ that is accepted by some IDPDA with $\Sigma_D = \{a\}$ and $\Sigma_R = \{b\}$. Any IDPDA with the same signature is clearly unable to accept the reversal $\{\, b^n a^n \mid n \geq 1 \,\}$. Thus, the family $\mathscr{L}$(IDPDA) is closed under reversal with arbitrary signatures, but not under reversal with identical signatures.

Lemma 5.4. *The family $\mathscr{L}$(DIDPDA) is neither closed under Kleene star nor under reversal.*

Proof. We consider again the language L used in the proof of Lemma 3.4, which is the concatenation of the language $L_5 = \{\, a^n b^m \mid n, m \geq 1 \text{ and } n < m \,\}$ with itself, where L_5 itself can be accepted by some DIDPDA.

Let us assume that $\mathscr{L}$(DIDPDA) is closed under Kleene star. Then, language $L_5^* \cap a^+b^+a^+b^+$ is accepted by some DIDPDA, since the intersection with the regular set $a^+b^+a^+b^+$ can be checked in the state set. However, $L_5^* \cap a^+b^+a^+b^+ = L$ which gives a contradiction, since L is not accepted by any DIDPDA.

For the non-closure under reversal, we consider the language

$$L^R = \{\, b^n a^m b^\ell a^k \mid 1 \leq k < \ell \text{ and } 1 \leq m < n \,\},$$

that can be accepted by a DIDPDA M with the signature $\Sigma_D = \{a\}$, $\Sigma_R = \{b\}$, and $\Sigma_N = \emptyset$ as follows. The correct format of the input can be checked using the state set of M. While reading the first b-block, a b-hole is dug. The following a-block fills this b-hole. As soon as the b-hole is completely filled, the input has to be rejected and a blocking non-accepting state is entered. If the b-hole has a depth of at least one, the computation continues and the following b-block digs a c-hole, where the first dug symbol is c'. Then, the final a-block fills this c-hole. The input is accepted if the symbol c' is not filled. As soon as the symbol c' is filled, a blocking non-accepting state is entered. Thus, M accepts if the first a-block is strictly shorter than the first b-block as well as the second a-block is strictly shorter than the second b-block.

Assume the family $\mathscr{L}$(DIDPDA) is closed under reversal. Then, $(L^R)^R = L$ is accepted by some DIDPDA. However, Lemma 3.4 shows that L is not accepted by any DIDPDA. This gives a contradiction and the desired non-closure result. □

Next, we will discuss some decidability questions for DIDPDAs. Let us recall that a decidability problem is *undecidable* whenever the set of all instances for which the answer is "yes" is not recursive, whereas a decidability problem is *semidecidable* whenever the set of all instances for which the answer is "yes" is recursively enumerable. The family $\mathscr{L}$(DIDPDA) is a subset of the deterministic context-free languages. Thus, all decidability questions that are decidable for DPDAs are decidable for DIDPDAs as well.

Theorem 5.5. *The problems of emptiness, universality, finiteness, infiniteness, equivalence, and regularity are decidable for DIDPDAs.*

It is known that the inclusion problem for deterministic context-free languages is undecidable. However, for DIDPDAs with compatible signatures it *is* decidable.

Theorem 5.6. *Let M and M' be two DIDPDAs with compatible signatures. Then, the question whether $L(M) \subseteq L(M')$ is decidable.*

Proof. The test of inclusion $L(M) \subseteq L(M')$ is equivalent to test $L(M) \cap \overline{L(M')}$ for emptiness. We know that the family $\mathscr{L}$(DIDPDA) is effectively closed under complementation by Lemma 5.2 and its proof shows that the signature of the constructed automaton for the complement is not changed. Moreover, the family $\mathscr{L}$(DIDPDA) is effectively closed under intersection with compatible signatures as well by Lemma 5.2. Since emptiness is decidable by Theorem 5.5, we obtain that the inclusion problem is decidable as well. □

The inclusion problem becomes undecidable and, moreover, not even semidecidable if the signatures are no longer compatible.

Theorem 5.7. *Let M and M' be two DIDPDAs. Then, the question whether $L(M) \subseteq L(M')$ is not semidecidable.*

Proof. It is shown in [13] that the inclusion problem is not semidecidable for two IDPDAs with not necessarily compatible signatures. Moreover, it can be assumed that the pushdown store of the two IDPDAs is in fact used as a counter only. The basic idea of the proof in [13] is to construct two IDPDAs M and M' such that the intersection $L(M) \cap L(M')$ represents the suitably encoded version of the valid computations of a counter machine. Then, the inclusion problem can be related with the emptiness problem of a counter machine which is known to be not semidecidable.

In principle, this approach can be chosen here again: taking a close look at the construction of M and M', one can observe that in case of acceptance M and M' never pop from an empty pushdown store. Hence, M and M' can be considered as DIDPDAs in this case as well. In case of rejection, we can modify M and M' analogously to the construction in the proof of Lemma 5.1 to enter a permanent non-accepting state s_- in case of situations with undefined transition function and situations where a pop operation from an empty pushdown store takes place. Hence, M and M' are DIDPDAs in the rejecting case as well. Altogether, we can construct DIDPDAs M and M' such that $L(M) \cap L(M')$ represents the valid computations of a counter machine. This shows again that the inclusion problem is not semidecidable. □

Finally, we look at the computational complexity of the decidable questions.

Theorem 5.8. *The emptiness problem for DIDPDAs and NDIDPDAs is* P*-complete. The universality, equivalence, and inclusion problems for DIDPDAs are* P*-complete. The universality, equivalence, and inclusion problems for NDIDPDAs are* EXPTIME*-complete.*

Proof. The emptiness problem for general (nondeterministic) pushdown automata is in P. This implies that the emptiness problem for DIDPDAs and NDIDPDAs is in P as well. Since the inclusion problem can be reduced to the emptiness problem due to the proof of Theorem 5.6 and the constructions involved are intersection and complementation which cause a polynomial blow-up only, the inclusion problem for DIDPDAs belongs to P. Hence, the universality and equivalence problems for DIDPDAs belong to P as well. The P-hardness of the emptiness problem for deterministic and nondeterministic IDPDAs is shown in [17] and is a reduction from the alternating graph reachability problem. A different proof is given in [22] which is a reduction from the monotone circuit value problem. The basic idea is to construct a deterministic IDPDA that accepts a unique string if and only if the given circuit evaluates to 1. Since the accepted string is well-nested with respect to the signature of the IDPDA, in the accepting computation there is no situation in which a pop operation on the empty pushdown takes place. Thus, the deterministic IDPDA is in fact a DIDPDA for the accepting computation. For the rejecting computations, we use the transitions of the given deterministic IDPDA, but enter, similar to the construction in the proof of Lemma 5.1, a new rejecting state that is never left, if a situation occurs in which a pop operation on the empty pushdown takes place. Hence, the emptiness problem for DIDPDAs and NDIDPDAs is P-hard as well. This gives also the P-hardness of the equivalence and inclusion problems for DIDPDAs, since for all problems one can choose to test the equivalence with the empty set and inclusion in the empty set, respectively. The P-hardness of the universality problem can be obtained similarly by taking into account that DIDPDAs are closed under complementation by Lemma 5.2.

The universality, equivalence, and inclusion problems for NDIDPDAs are in EXPTIME, since by Theorem 4.2 every NDIDPDA can be converted into an equivalent DIDPDA with exponential blow-up and then the problems considered can be solved for DIDPDAs as described above. The EXPTIME-hardness of universality is shown in [1] (see also [22]) for nondeterministic IDPDAs by a reduction from the membership problem for alternating linear-space Turing machines. The basic idea is to construct a nondeterministic IDPDA M' for a given Turing machine M and an input w such that $L(M') = \Sigma^*$ if and only if $w \notin L(M)$. In detail, the automaton M' accepts every input but valid encodings of accepting computation trees of M on w. This is realized by guessing a position while reading the input string and to check whether an error really occurs that makes the encoding invalid. Only in this case, the input is accepted. The pushdown store of the IDPDA is basically used to check whether a part of the input string is not of the form xyx^R for non-empty strings x and y. Now, we have to ensure that this behavior can also be realized by an NDIDPDA M''. The guessing and checking of an error can be realized in the same way as in the IDPDA M'. However, we have to ensure that the signature of the input is obeyed before and after the guessing and checking phase and that the guessing and checking phase does not start with a hole on the pushdown store. The first property is obtained by applying again a similar construction as in the proof of Lemma 5.1. We observe that a valid encoding of an accepting computation tree is a well-nested string. This means that as soon as M'' pops from the empty pushdown store, we can enter an accepting state s_+ which is never left until the complete input is read. This ensures also the second property: if the guessing and checking phase starts, then we know that there is no hole on the pushdown store and we can proceed as in M'. Whenever in the following computation an error is detected or M'' pops from the empty pushdown store, we enter the accepting state s_+. Thus, we obtain the EXPTIME-hardness of universality for NDIDPDAs in a similar way as for nondeterministic IDPDAs. This gives also the EXPTIME-hardness of the equivalence and inclusion problem, since we can test the equivalence with Σ^* or the inclusion of Σ^*. □

6. On the union of the families accepted by DIDPDAs and IDPDAs

The preceding sections showed that DIDPDAs as well as IDPDAs are interesting special cases of deterministic pushdown automata, since both classes have strong closure properties as well as the decidability of inclusion in case of compatible signatures which is in contrast to general deterministic pushdown automata. Furthermore, it has been shown in Theorem 3.5 that the language families $\mathscr{L}(\text{DIDPDA})$ and $\mathscr{L}(\text{IDPDA})$ are incomparable. Thus, it is natural to consider the union of both language families, that is, the family of languages which are accepted by DIDPDAs or IDPDAs, and to ask whether the new language family shares the closure properties as well as the decidability of inclusion with its subfamilies. In this section, we will investigate these questions and it turns out that the new family is no longer closed under union and intersection with compatible signatures. Moreover, the inclusion problem for two automata becomes non-semidecidable even if both automata have the same signature. This is in strong contrast to the results known for DIDPDAs as well as for IDPDAs.

Let us denote by $\mathscr{L}$ the family of languages which are accepted by DIDPDAs or IDPDAs. Formally, we define $\mathscr{L} = \mathscr{L}(\text{DIDPDA}) \cup \mathscr{L}(\text{IDPDA})$. Since the families $\mathscr{L}(\text{DIDPDA})$ and $\mathscr{L}(\text{IDPDA})$ are incomparable, it is clear that both families are proper subsets of $\mathscr{L}$. Moreover, $\mathscr{L}$ is properly included in the family of real-time deterministic context-free languages. A witness is again language $\{\, a^n\$a^n \mid n \geq 0 \,\}$ that is neither accepted by any DIDPDA nor by any IDPDA.

Considering the closure properties of $\mathscr{L}$ we have the following positive closure results.

Lemma 6.1. *The language family $\mathscr{L}$ is closed under complementation and under union and intersection with regular languages.*

Proof. Any language from $\mathscr{L}$ is either accepted by a DIDPDA or an IDPDA, say M. For the closure under union and intersection with regular languages it suffices to extend the state set of M by another component in which the deterministic finite automaton representing the regular language given is simulated. The closure under complementation follows from the fact that both $\mathscr{L}(\text{DIDPDA})$ and $\mathscr{L}(\text{IDPDA})$ are closed under complementation (see Lem. 5.2 and [1]). □

Next, we turn to non-closure results.

Lemma 6.2. *The family $\mathscr{L}$ is not closed under union, intersection, and concatenation even with compatible signatures. It is neither closed closed under Kleene star, reversal, inverse homomorphism nor under length-preserving homomorphism.*

Proof. We consider $L_3 = \{ a^n b^n c^m \mid n, m \geq 1 \}$ and $L_4 = \{ a^n b^m c^m \mid n, m \geq 1 \}$ that have already been used in the proof of Lemma 5.3. Language L_4 can be accepted by an IDPDA with the signature $\Sigma_D = \{b\}$, $\Sigma_R = \{a, c\}$, and $\Sigma_N = \emptyset$, while L_3 is accepted by a DIDPDA with the same signature. Thus, $L_3 \cap L_4$ belongs to $\mathscr{L}$, if $\mathscr{L}$ is closed under intersection with compatible signatures. Since $L_3 \cap L_4$ is a non-context-free language, we obtain the non-closure under intersection with compatible signatures. Since $\mathscr{L}$ is closed under complementation by Lemma 6.1, the non-closure under union with compatible signatures can be derived as well. Next, we consider language $L_5 = \{ a^n b^m \mid n, m \geq 1 \text{ and } n < m \}$ that is accepted by some IDPDA with the signature $\Sigma_D = \{a\}$, $\Sigma_R = \{b\}$, and $\Sigma_N = \emptyset$ as well as the language $L_6 = \{ w \in \{a, b\}^+ \mid |w|_a < |w|_b \}$ that is accepted by some DIDPDA with the same signature. Now, assume that $\mathscr{L}$ is closed under concatenation. Then, $L_5 \cdot L_6$ belongs to $\mathscr{L}$. If $L_5 \cdot L_6$ is accepted by some DIDPDA, then the intersection $(L_5 \cdot L_6) \cap a^+b^+a^+b^+$ is accepted by a DIDPDA as well. However, this intersection gives the language $L = \{ a^k b^\ell a^m b^n \mid 1 \leq k < \ell \text{ and } 1 \leq m < n \}$ used in the proof of Lemma 3.4 and it is known that L is not accepted by any DIDPDA. Thus, $L_5 \cdot L_6$ has to be accepted by an IDPDA. However, a straightforward proof shows that $L_5 \cdot L_6$ cannot be accepted by any IDPDA as well. This gives the contradiction and the non-closure under concatenation with compatible signatures.

Next, we assume that $\mathscr{L}$ is closed under Kleene star. We define the language $L_7 = L_5 \cup cL_6$ and observe that L_7 is accepted by some DIDPDA and, hence, belongs to $\mathscr{L}$. Since $\mathscr{L}$ is closed under Kleene star, language L_7^* belongs to $\mathscr{L}$ as well and is accepted by some DIDPDA or IDPDA. If L_7^* is accepted by some DIDPDA, then $L_7^* \cap a^+b^+ca^+b^+$ is accepted by some DIDPDA as well. However, $L_7^* \cap a^+b^+ca^+b^+ = \{ a^k b^\ell c a^m b^n \mid 1 \leq k < \ell \text{ and } 1 \leq m < n \} = L'$ which is basically the language L used in the proof of Lemma 3.4. Then, a proof almost identical to the proof of Lemma 3.4 shows that L' is not accepted by any DIDPDA which is a contradiction. Thus, L_7^* is accepted by some IDPDA. Hence, the language $L_7^* \cap c\{a, b\}^+ = cL_6$ is accepted by an IDPDA as well. Again, a straightforward proof shows that cL_6 cannot be accepted by any IDPDA which gives the contradiction and the non-closure under Kleene star.

Now, we assume that $\mathscr{L}$ is closed under reversal. We define the language $L_8 = (L_5cL_5)^R \cup cL_6$ and note that L_8 is accepted by some DIDPDA, since $(L_5cL_5)^R$ can be accepted by some DIDPDA similar to the construction given in the proof of Lemma 5.4. Hence, L_8 belongs to $\mathscr{L}$. Since $\mathscr{L}$ is closed under reversal, language L_8^R belongs to $\mathscr{L}$ as well and is accepted by some DIDPDA or IDPDA. If L_8^R is accepted by some DIDPDA, then we consider $L_8^R \cap a^+b^+ca^+b^+$ and obtain again language L' which is not accepted by any DIDPDA. Thus, L_8^R is accepted by an IDPDA which implies that $L_8^R \cap \{a, b\}^+c = L_6c$ is accepted by an IDPDA as well. Again, a straightforward proof shows that L_6c cannot be accepted by any IDPDA which gives the contradiction and the non-closure under reversal.

The proofs for the non-closure under length-preserving homomorphism and inverse homomorphism are identical to the proofs given for Lemma 5.3: The languages $L_1 = \{ a^n \$ a^n \mid n \geq 1 \}$ and $L_2 = \{ a^n \$ b^{2n} \mid n \geq 1 \}$ that are each not accepted by any DIDPDA or IDPDA are obtained as length-preserving homomorphic image and inverse homomorphic image of the languages $L_1' = \{ a^n \$ b^n \mid n \geq 1 \}$ and $L_2' = \{ a^{2n} \$ b^{2n} \mid n \geq 1 \}$, respectively, which in turn are each accepted by some DIDPDA or IDPDA. □

All results on the closure properties of Section 5 and Section 6 are summarized in Table 1.

The inclusion problem is known to be solvable in polynomial time if both automata have compatible signatures and, moreover, are both DIDPDAs or are both IDPDAs. Our next results shows that this situation changes drastically if we consider the inclusion problem for two automata with the same signature, but one automaton being a DIDPDA and the other one being an IDPDA. In this situation, it turns out that the inclusion problem becomes undecidable and, moreover, not even semidecidable.

The non-semidecidability of the inclusion problem for automata from $\mathscr{L}$ that even have identical signatures is shown by reduction of the emptiness problem for deterministic linearly space bounded one-tape, one-head Turing machines, so-called linear bounded automata (LBA). We remark that the non-semidecidability of the latter problem may be obtained by applying the result shown in [23] that every recursively enumerable language L accepted by some Turing machine can be represented as the homomorphic image $h(L')$ of a context-sensitive language L' accepted by some LBA. Since $L = h(L')$ is empty if and only L' is empty, the emptiness problem for Turing machines, which is known to be not semidecidable, is reduced to the emptiness problem for LBAs.

TABLE 1. Closure properties of the language families discussed. Symbols $\cup_c$, $\cap_c$, and $\cdot_c$ denote union, intersection, and concatenation with compatible signatures. Such operations are not defined for DFAs and DPDAs and they are marked with '—'. The abbreviations $h_{l.p.}$ and h^{-1} denote length-preserving homomorphism and inverse homomorphism.

	$\overline{}$	$\cup$	$\cap$	$\cup_c$	$\cap_c$	$\cdot$	$\cdot_c$	$*$	$h_{l.p.}$	h^{-1}	R
DFA	✓	✓	✓	—	—	✓	—	✓	✓	✓	✓
IDPDA	✓	✗	✗	✓	✓	✗	✓	✓	✗	✗	✓
DIDPDA	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗
$\mathscr{L}$	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
DPDA	✓	✗	✗	—	—	✗	—	✗	✗	✓	✗

For the reduction of the emptiness problem for LBAs to the inclusion problem for automata from $\mathscr{L}$ with identical signatures we encode histories of LBA computations in single words that are called *valid computations* (see, for example, [10]). We may assume that LBAs get their input in between two endmarkers, make no stationary moves, accept by halting in some unique state f on the leftmost input symbol after an even number of steps, and are sweeping, that is, the read-write head changes its direction at endmarkers only. Let δ be the transition function of some LBA M and Q be its state set, where q_0 is the initial state, T with $T \cap Q = \emptyset$ is the tape alphabet containing the endmarkers $\triangleright$ and $\triangleleft$, and $\Sigma \subset T$ is the input alphabet. Since M is sweeping, the set of states can be partitioned into Q_R and Q_L of states appearing in right-to-left and in left-to-right moves. A configuration of M can be written as a string of the form $\triangleright T^*QT^*\triangleleft$ such that, $\triangleright t_1t_2\cdots t_ist_{i+1}\cdots t_n\triangleleft$ is used to express that M is in state s, $\triangleright t_1t_2\cdots t_n\triangleleft$ is the tape inscription, for $s \in Q_R$ tape symbol t_{i+1} is scanned, and for $s \in Q_L$ tape symbol t_i is scanned. Now, we consider words of the form $w_0w_1^Rw_2\cdots w_{2m-1}^Rw_{2m}$, where $w_i \in \{\triangleright\}T^*QT^*\{\triangleleft\}$ are configurations of M, w_0 is an initial configuration of the form $\triangleright q_0\Sigma^*\triangleleft$, $w_{2m} \in \{\triangleright f\}T^*\{\triangleleft\}$ is a halting, that is, accepting configuration, and w_{i+1} is the successor configuration of w_i. These words are encoded so that every state symbol is merged together with its both adjacent symbols into a metasymbol. To this end, we assume that the LBA input is nonempty, and rewrite every substring of $w_0w_1^Rw_2\cdots w_{2m-1}^Rw_{2m}$ having the form tqt' to $[t,q,t']$, where $q \in Q$ and $t,t' \in T$. Additionally, we consider copies $\overline{Q}$ and $\overline{T}$ of the encoding alphabet and use for every reversed configuration the overlined alphabet. The set of these encodings is defined to be the set of valid computations of M. We denote it by VALC(M).

Example 6.3. We consider the following computation of an LBA on input $x_1x_2x_3$ where each configuration consists of the current tape inscription, the current state, and the current position of the read-write head, $q_0,\dots,q_3,f \in Q_R$ and $p_0,p_1,p_2,p_3 \in Q_L$.

1: $(\triangleright x_1x_2x_3\triangleleft, q_0, 1)$	4: $(\triangleright y_1y_2y_3\triangleleft, q_3, 4)$	7: $(\triangleright y_1z_2z_3\triangleleft, p_1, 1)$
2: $(\triangleright y_1x_2x_3\triangleleft, q_1, 2)$	5: $(\triangleright y_1y_2y_3\triangleleft, p_3, 3)$	8: $(\triangleright z_1z_2z_3\triangleleft, p_0, 0)$
3: $(\triangleright y_1y_2x_3\triangleleft, q_2, 3)$	6: $(\triangleright y_1y_2z_3\triangleleft, p_2, 2)$	9: $(\triangleright z_1z_2z_3\triangleleft, f, 1)$

The corresponding valid computation is:

$$[\triangleright, q_0, x_1]x_2x_3\triangleleft\overline{\triangleleft}\overline{x}_3[\overline{y}_1,\overline{q}_1,\overline{x}_2]\overline{\triangleright}\triangleright y_1[y_2,q_2,x_3]\triangleleft[\overline{y}_3,\overline{q}_3,\overline{\triangleleft}]\overline{y}_2\overline{y}_1\overline{\triangleright}\triangleright y_1y_2[y_3,p_3,\triangleleft]$$

$$\overline{\triangleleft}[\overline{y}_2,\overline{p}_2,\overline{z}_3]\overline{y}_1\overline{\triangleright}\triangleright[y_1,p_1,z_2]z_3\triangleleft\overline{\triangleleft}\overline{z}_3\overline{z}_2[\overline{\triangleright},\overline{p}_0,\overline{z}_1][\triangleright,f,z_1]z_2z_3\triangleleft$$

Lemma 6.4. *Let M be an LBA. Then an IDPDA M_1 and a DIDPDA M_2 both having the same signature can effectively be constructed such that $L(M_1) \cap L(M_2) = VALC(M)$.*

Proof. Let us first describe an IDPDA M_1 with signature $\Sigma_D = \overline{Q} \cup \overline{T} \cup (\overline{T} \times \overline{Q} \times \overline{T})$, $\Sigma_R = Q \cup T \cup (T \times Q \times T)$, and $\Sigma_N = \emptyset$: First of all, the state set of M_1 is used to check the correct format of the input. This means that the first configuration is an initial configuration, the last configuration is an accepting configuration, the alphabet for the configurations alternates between non-overlined and overlined symbols, and that every configuration itself is of a correct format. It should be noted that due to the definition of VALC(M) every configuration has the same length and any two adjacent configurations differ on two adjacent positions only disregarding the overlining of symbols and the merging into metasymbols. Now, the main task for M_1 is to check starting with the third configuration whether every non-overlined configuration is the successor configuration of its preceding overlined configuration. To this end, M_1 reads and ignores the first configuration while its pushdown store remains empty. Then, every next configuration (with overlined symbols) is read and every symbol is pushed onto the pushdown store. Then, the following configuration (with non-overlined symbols) is read and checked against the pushdown store. If a metasymbol of the form $[t, q, t']$ is popped off or read in the input, M_1 checks within the next two transitions by using its state set whether the two input symbols encode a correct part of the successor configuration based on the two symbols popped off the pushdown store. If the check is positive, the computation continues. Otherwise, the computation ends non-accepting. If neither the popped symbol nor the input symbol read is a metasymbol and both symbols are equal, the computation continues. In case of non-equivalence, the computation ends non-accepting. Since all configurations have to have the same length, any pop operation on the empty pushdown store after the first configuration leads to a non-accepting computation end as well. Otherwise, the check of the next two configurations is started. In this way, the input is accepted if it is correctly formatted and every non-overlined configuration is the successor configuration of its preceding overlined configuration.

The construction of a DIDPDA M_2 with the same signature is similar, since it has to be checked whether every overlined configuration is the successor configuration of its preceding non-overlined configuration. To this end, the first configuration is read which digs a hole of the shape of the configuration read. The next overlined configuration is read and checked against the hole on the pushdown store as follows: If a metasymbol of the form $[t, q, t']$ is seen at the bottom of the hole in the pushdown store or read in the input, M_2 checks within the next two transitions by using its state set whether the two input symbols encode a correct part of the successor configuration based on the two symbols which are pushed into the hole in the pushdown store. If the check is positive, the computation continues. Otherwise, the computation ends non-accepting. If neither the symbol seen at the bottom of the hole in the pushdown store nor the input symbol read is a metasymbol and both symbols are equal, the computation continues by pushing the adequate symbol into the hole in the pushdown store. In case of non-equivalence, the computation ends non-accepting. Since all configurations have to have the same length, any push operation on the empty pushdown store leads to a non-accepting computation end as well. Otherwise, the check of the next two configurations is started. In this way, the input is accepted if every overlined configuration is the successor configuration of its preceding non-overlined configuration.

Altogether, the inputs that are accepted by M_1 and M_2 are those strings which are correctly formatted and where every non-overlined configuration is the successor configuration of its preceding overlined configuration as well as every overlined configuration is the successor configuration of its preceding non-overlined configuration, hence a valid computation of the given LBA. □

Theorem 6.5. *Let M be an IDPDA and M' be a DIDPDA with the same signature. Then, the inclusion $L(M) \subseteq L(M')$ is not semidecidable.*

Proof. Due to Lemma 6.4 we know how to construct an IDPDA M_1 as well as a DIDPDA M_2 with the same signature such that the intersection $L(M_1) \cap L(M_2)$ represents the valid computations of an arbitrary LBA. Due to Lemma 5.2 a DIDPDA M' with the same signature as M_2 can be constructed that accepts the complement $\overline{L(M_2)}$. Now, assume that the inclusion problem is semidecidable. Then, $L(M_1) \subseteq L(M')$ is semidecidable which is equivalent to semidecide whether $L(M_1) \cap \overline{L(M')} = L(M_1) \cap L(M_2) = \emptyset$. Hence, the emptiness of the valid computations of an LBA as well as the emptiness problem for LBAs would be semidecidable which is a contradiction. □

References

[1] R. Alur and P. Madhusudan, Visibly pushdown languages, in *Symposium on Theory of Computing (STOC 2004)*. ACM (2004) 202–211.

[2] R. Alur and P. Madhusudan, Adding nesting structure to words. *J. ACM* **56** (2009).

[3] S. Bensch, M. Holzer, M. Kutrib and A. Malcher. Input-driven stack automata. In *Theoretical Computer Science (TCS 2012)*, volume 7604 of *LNCS*. Springer (2012) 28–42.

[4] D. Carotenuto, A. Murano and A. Peron, Ordered multi-stack visibly pushdown automata. *Theoret. Comput. Sci.* **656** (2016) 1–26.

[5] P. Chervet and I. Walukiewicz, Minimizing variants of visibly pushdown automata. In *Mathematical Foundations of Computer Science (MFCS 2007)*, volume 4708 of *LNCS*. Springer (2007) 135–146.

[6] S. Crespi-Reghizzi and D. Mandrioli, Operator precedence and the visibly pushdown property. *J. Comput. System Sci.* **78** (2012) 1837–1867.

[7] P.W. Dymond, Input-driven languages are in $\log n$ depth. *Inform. Process. Lett.* **26** (1988) 247–250.

[8] J. Goldstine, J.K. Price and D. Wotschke, On reducing the number of states in a PDA. *Math. Syst. Theory* **15** (1982) 315–321.

[9] J. Goldstine, J.K. Price and D. Wotschke, On reducing the number of stack symbols in a PDA. *Math. Syst. Theory* **26** (1993) 313–326.

[10] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979).

[11] M. Kutrib and A. Malcher, Digging input-driven pushdown automata. In *Eleventh Workshop on Non-Classical Models of Automata and Applications, NCMA 2019, Valencia, Spain, July 2-3, 2019*. Österreichische Computer Gesellschaft (2019) 109–124.

[12] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano and M. Wendlandt, Deterministic input-driven queue automata: finite turns, decidability, and closure properties. *Theoret. Comput. Sci.* **578** (2015) 58–71.

[13] M. Kutrib, A. Malcher and M. Wendlandt, Tinput-driven pushdown, counter, and stack automata. *Fund. Inf.* **155** (2017) 59–88.

[14] S. La Torre, P. Madhusudan and G. Parlato, A robust class of context-sensitive languages. In *Logic in Computer Science (LICS 2007)*. IEEE Computer Society (2007) 161–170.

[15] S. La Torre, M. Napoli and G. Parlato, On multi-stack visibly pushdown languages. Preprint (2013). http://eprints.soton.ac.uk/id/eprint/351914.

[16] S. La Torre, M. Napoli and G. Parlato, Scope-bounded pushdown languages. *Int. J. Found. Comput. Sci.* **27** (2016) 215–234.

[17] M. Lange, P-hardness of the emptiness problem for visibly pushdown languages. *Inform. Process. Lett.* **111** (2011) 338–341.

[18] P. Madhusudan and G. Parlato, The tree width of auxiliary storage. In *Principles of Programming Languages, (POPL 2011)*. ACM (2011) 283–294.

[19] A. Meduna and P. Zemek, Jumping finite automata. *Int. J. Found. Comput. Sci.* **23** (2012) 1555–1578.

[20] K. Mehlhorn, Pebbling moutain ranges and its application of DCFL-recognition. In *International Colloquium on Automata, Languages and Programming (ICALP 1980)*. Volume 85 of *LNCS*. Springer (1980) 422–435.

[21] B. Nagy and F. Otto, Finite-state acceptors with translucent letters. In *International Workshop on AI Methods for Interdisciplinary Research in Language and Biology (ICAART 2011)*. INSTICC, SciTePress (2011) 3–13.

[22] A. Okhotin and K. Salomaa, Complexity of input-driven pushdown automata. *SIGACT News* **45** (2014) 47–67.

[23] A. Salomaa, Formal Languages. Academic Press (1973).

[24] B. von Braunmühl and R. Verbeek, Input-driven languages are recognized in $\log n$ space. In *Topics in the Theory of Computation*. Volume 102 of Mathematics Studies. North-Holland, Amsterdam (1985) 1–19.